CHALMERS
UNIVERSITY OF TECHNOLOGY

Testing and validation of safety logic in the virtual environment

Citation for the original published paper (version of record):
Khan, A., Falkman, P., Fabian, M. (2019). Testing and validation of safety logic in the virtual environment. CIRP Journal of Manufacturing Science and Technology, 26: 1-9.
http://dx.doi.org/10.1016/j.cirpj.2019.07.002

CIRP Journal of Manufacturing Science and Technology 26 (2019) 1–9

Contents lists available at ScienceDirect

CIRP Journal of Manufacturing Science and Technology

journal homepage: www.elsevier.com/locate/cirpj

Testing and validation of safety logic in the virtual environment

Adnan Khan*, Petter Falkman, Martin Fabian

Chalmers University of Technology, Department of Electrical Engineering, 41296 Göteborg, Sweden

ARTICLE INFO

Article history:
Received 7 October 2018
Accepted 23 July 2019
Available online 10 August 2019

Keywords:
Virtual preparation
Virtual commissioning
Simulation model
Safety logic
PLC logic

ABSTRACT

This paper presents an approach for testing safety PLC logic in a virtual environment, using the IOCO testing relation as validation criteria. Manufacturing companies more and more rely on virtual commissioning to reduce the physical commissioning time by testing and debugging the PLC logic of the nominal behavior prior to physical commissioning. However, safety PLC logic testing is still carried out on real systems manually. This manual practice of safety logic validation hinders industry to exploit the full potential of virtual commissioning to reduce the physical commissioning time. The proposed approach assists manufacturing companies in the validation of safety PLC logic using a simulation model before the *factory acceptance testing* phase. Using the proposed approach, a simulation model can be used to test the safety PLC logic and prepare better for the factory acceptance testing phase, hence, further reduction in physical commissioning time can be achieved.

1 Introduction

Manufacturing companies opt for more automated solutions by introducing robots and other automated machines, which has positively affected the production rate and product quality. To achieve an automated solution, these machines are controlled by PLCs (Programmable Logic Controllers), which are typically manually programmed and thus prone to human errors. The errors found in the programs are corrected during the physical commissioning of the production system, hence causing delays in the commissioning of the production system. According to [1], about seventy percent of the physical commissioning time is consumed in correcting errors of the PLC code.

To counter this problem, *virtual commissioning* is being used by several manufacturing companies prior to physical commissioning. For virtual commissioning, simulation and modeling engineers first create a (complete) simulation model of a production system. The specification documents that are used to create the simulation model are a combination of formal and informal descriptions.

After modeling, the PLC code related to the nominal behavior of different machines is created and tested using the simulation model. The testing and validation of the PLC code helps in reducing the physical commissioning time [2,3], due to rectification of errors before physical commissioning.

In addition to reduced physical commissioning time [2,3], a robust simulation model is gained as a byproduct, which can be used to test future modifications. Testing modifications initially in the virtual environment helps in determining the behavior of a production system in a safe way, as modifications made directly on a physical system might cause production disruption and or equipment damage.

Many different concepts and approaches to virtual commissioning exist [4–7], and have shown great promise when it comes to reducing the physical commissioning time. However, testing and validation of the safety PLC code is typically still carried out on the real production system. To harvest the full potential of virtual commissioning, testing and validation of the safety PLC logic in a virtual environment using a simulation model warrants further investigation.

In many companies, the engineering staff is not involved in creating the PLC code. Instead, third-party contractors create the PLC code including the safety PLC code. Therefore, it is difficult to confirm anything until the factory acceptance testing is carried out. On many occasions, even after the factory acceptance testing, errors can still be found in the PLC code. These errors are often corrected after the delivery of the production cell, hence adding more time to the physical commissioning.

Due to the problems associated with the third party validation procedure, the safety PLC code validation procedure needs amendment so that the physical commissioning time can be further reduced. Also, in order to ensure that the production cell conforms to the required specifications, the testing and validation procedure should be made transparent for the manufacturing

* Corresponding author.
E-mail addresses: adnan.khan@chalmers.se (A. Khan), petter.falkman@chalmers.se (P. Falkman), fabian@chalmers.se (M. Fabian).

https://doi.org/10.1016/j.cirpj.2019.07.002
1755-5817/

company personnel. The current validation procedure performed during the factory acceptance test is based solely on visual inspection using checklists, which requires days and sometimes weeks at the contractor's location [8].

The checklists used by the manufacturing company during the factory acceptance test are expressed in natural language. Natural language descriptions can lead to false validation of PLC code as it can be interpreted differently. Also, only assessing behavioral equivalence by visual inspection is not enough, since minor but important errors can be overlooked by the engineers. The visual inspection procedure can be strengthened by formally testing safety properties.

In [9] is discussed the importance of formal specifications for safety PLC program verification but not testing and validation. Validating certain safety aspects based on formally specified tests will result in a more robust PLC code as the errors related to the use of natural language can be avoided. The use of natural language apart from interpretation issues also limits the computer algorithms to auto-generate tests.

Model-based testing [10] is an approach in which a model of an implementation undergoes a series of tests to find errors. If the implementation conforms to the specification, the test is passed, else it fails. In the case of safety PLC code, the safety code will be the implementation which undergoes testing. There are two ways to carry out model-based testing, offline or online.

In online testing, tests are developed and run during the test execution on a system, whereas in offline testing, test cases are developed before and are executed later on the system under test.

In offline model-based testing, issues of state-space explosion and handling of non-determinism are intrinsic [11]. But to test safety PLC logic in a virtual environment, offline model-based testing is the most suited option as the production system is not yet commissioned physically. Also, the concept of virtual commissioning revolves around a simulation model, hence online testing would not fit in this case.

The literature does not, to the best of the authors' knowledge, include any approach that incorporates offline model-based testing for validation of safety PLC logic using a simulation model.

1.1 Contribution

In this paper, an approach to test and validate the safety PLC logic based on model-based testing is presented. The proposed setup, illustrated in Fig. 1 consists of a safety PLC, a standard PLC, a test initiation software, and a simulation model. The tests are initiated in the simulation model via the test initiation software, which also makes the decision about whether the executed test has either passed or failed. After the initiation of a test, the sequence of nominal actions is disrupted in the simulation model, which updates the input–output values in both the standard PLC and the safety PLC. Based on this new status of the input–output from the safety PLC, the test initiation software determines if the executed test passed or failed. The approach is aimed to support safety PLC logic validation before the factory acceptance testing phase to further cut down physical commissioning time. Using the proposed approach, the engineers will be more prepared and probably spend less time during factory acceptance testing, due to prior testing of the safety PLC logic in the virtual environment.

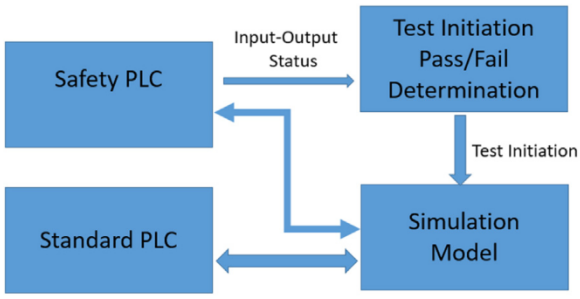

Fig. 1. Proposed setup.

1.2 Outline

This paper is structured in the following way. In Section 2, virtual commissioning practices are briefly described, together with some future possibilities. In Section 3, the background of industrial practice regarding PLC safety logic validation is detailed. Section 4 details the model-based testing approach. Section 5 presents the proposed approach for safety logic validation in a virtual environment. Section 6 concludes the paper with some potential future work directions.

2 Virtual commissioning

The concept virtual commissioning was introduced to cut down physical commissioning time [2,3]. There are three key elements essential for a virtual commissioning setup: a *simulation model* of the production system; a *programmable logic controller* (PLC) that interacts with and controls the simulation model; and an *emulator* that interconnects the PLC and the simulation model.

To build a simulation model, different proprietary software such as Process Simulate [12], Delmia [13] etc., can be used. Typically, simulation models are built to the level of sensors and actuators [14], the idea is to capture the behavior of a production system's relevant parts that would be appropriate for PLC control. The modeling details may vary depending on the expertise and requirements of the company. The goal is to fix the PLC code errors before the physical commissioning, so that the time spent during real commissioning can be cut down.

The PLC is a device that runs a program to control a physical production system. The PLC program can be broadly classified into two categories: PLC code for *nominal aspects* e.g. welding, gluing etc., and PLC code for *safety aspects* e.g. emergency stops, collision detection etc. Typically for virtual commissioning, companies implement and test PLC code related to nominal aspects with an assumption that the simulation model built in the first step of virtual commissioning exhibits the actual behavior. Based on this assumption, developing and testing the PLC code against the model allows the engineers to find and correct errors.

The emulator is required to establish a connection between the PLC and the simulation model, since it is not possible to connect inputs and outputs physically to a PLC during virtual commissioning due to the absence of real devices. The solution comes in the shape of an emulator e.g. Simba-box. After setting up the connections via the emulator, mapping of the signals is carried out. Details regarding setting up the connections, mapping the signals, etc., are given in [15].

In the typical approach of virtual commissioning, the complete PLC code is tested on a simulation model first and then the tested PLC code is implemented directly on a production system. Another approach, *Hybrid commissioning* [16], deals with a step-wise commissioning of the physical system. Hybrid commissioning is a configuration in which a simulation model and a physical system are simultaneously connected to a PLC. The idea is to commission the physical system in steps, rather than commissioning all equipment simultaneously.

For hybrid commissioning, first the simulation model is tested with the PLC and then each component, i.e. actuator and sensor, of the simulation model is replaced by its real counterpart [17]. This step-wise testing of different sub-systems of the physical

production system in a hybrid manner is useful in terms of finding control logic errors specific to a sub-system. Also, it is a safe practice compared to conventional virtual commissioning, because the complete production systems is not commissioned as a whole.

Another concept of using virtual commissioning during the entire preparation and commissioning phase as opposed to use virtual commissioning as a last step before physical commissioning of the project life cycle is proposed in [5–7]. By doing so, both the simulation model and the PLC logic can be tested at any stage of the engineering cycle. Also, having a single simulation model in the entire engineering chain would make the engineers' job easier in terms of modifying and updating the model at any stage of the engineering life cycle. The problem to counter here is data heterogeneity, as different proprietary software are used throughout the project life-cycle.

The problems of natural language and data heterogeneity can be resolved by specifying technical documents formally. A formal description can be translated by computer algorithms and can thus make the job of creating a simulation model and PLC code automatic. In industry, the use of formal methods is not routine but it is gradually gaining popularity. A number of works [18–22] motivate the use of formal approaches for both modeling and PLC programming.

The simulation model built using formal specification will allow information re-use from different software and tools used during the development phase. By reusing information and data, virtual commissioning and testing can start already in the preparation phase. Logical errors in the control logic can be found much earlier and the simulation model can be improved based on the results [7].

Currently, all the practices related to virtual commissioning whether it be creating a simulation model or creating PLC code are manual in nature. This manual implementation can be made fully automatic if ready-made models of components can be provided by vendors. A framework related to such automatic implementation is described in [23], but as of today, vendors do not provide ready-made components for modeling and PLC programming.

On the one hand, virtual commissioning has benefits in terms of cutting down physical commissioning time [2,3]. On the other hand, significant time is consumed to model the system down to the level of sensors and actuators [14]. Also, having a virtual environment gives leverage to test numerous scenarios, but it also requires more time to validate them. If safety PLC logic testing and validation is incorporated in the virtual environment, full potential of the virtual commissioning practice can be exploited, hence further reduction in physical commissioning time is possible.

3 Industrial practice of testing and validation of safety logic

When a new production system is installed, one of the last activities carried out during the factory acceptance test is the validation of the *safety system*. Safety systems are used primarily to protect humans from harm, but also to safe guard machines. There are different types of safety devices, e.g. light curtains, floor sensors, emergency switches etc., which are used to protect both humans and machines.

During factory acceptance testing, all the functions related to the safety equipment are tested manually, e.g. emergency stop buttons are pressed to see if the production system stops, light curtain functions are tested by manually breaking the light barrier etc. This manual testing of actual sensors and equipment triggers the associated inputs and outputs in the safety PLC code, the observed results helps in validation of the created safety PLC code.

The testing and validation of the safety system and safety logic is typically done manually using checklists and visual inspection [8]. The checklists are prepared in detail in conjunction with the equipment to be tested, and are typically described using natural language. Typically, it has titles of different tests, under the title comes the description of the test i.e. how the test is performed and what constitutes a pass for the executed text. For example, in the case of testing safety PLC logic for the emergency shutdown, the physical system will be interrupted while nominal operation is being carried out by pressing the button. This interruption will trigger the safety interlock in the PLC logic, which automatically takes the physical system from being operational to a halt state. After visually observing and confirming that machines and humans are safe, this shutdown logic will constitute a pass.

The checklists are managed by the *safety personnel* at the company level. The safety personnel is one or more authorized persons from one of the engineering departments within the company. The responsibilities of the safety personnel include maintaining safety documents related to validation e.g. checklists, development process (V-model) and equipment safety buy-offs.

A *certified engineer* is assigned by the lead engineer to provide technical support and to perform the validation of the safety software and safety devices. The certified engineer can be third party personnel hired to give support, or the company's own employee. The assigned engineer must have sound knowledge of control engineering, the company's hardware and software standards and should be experienced in validation of safety systems. The certified engineer validates the following aspects of the safety code:

- Completeness and accuracy.
- Implementation of predefined "standard non-editable routines".
- Correctness of I/O mapping and configurations.
- Implementation of correct interlocks.
- Exclusive implementation of safety tasks and routines from nominal tasks.

In addition to the above mentioned tasks, additional functional and dynamic tests based on design documentation, regional control organization norms and regulations are also carried out. After the validation task, a certificate of correct and successful validation is provided to the company.

The PLC programs are created based on the requirement specifications provided by the manufacturing company, which requires thorough investigation and testing, especially if the PLC programming is outsourced. Because when the PLC code is developed in-house at the company instead of outsourced to a contractor, the communication between the engineers who create the PLC code and the engineers who wrote the specification can take place anytime during the project life-cycle. This communication gap increases when the project is outsourced to another company, and has the consequence that the PLC code becomes developed from rather high level requirement specifications and probably requires more thorough testing.

Informal specifications can be interpreted differently, which leads to more errors, therefore the finished PLC code, irrespective of if it has been developed in-house or not, has to be tested. Currently in the preparation phase, usually the engineers at the manufacturing companies do robot simulation and other forms of modeling e.g. CAD and kinematics. But in terms of preparation for factory acceptance testing, manual checklists are prepared.

4 Model-based testing

Manufacturing production systems and their logical behavior can often be beneficially modeled as *discrete event systems* [24]. Discrete event systems evolve dynamically on the occurrence of *events*, while at each time instant occupying a specific *state* where certain conditions hold. Several discrete event formalisms and approaches [24] have been developed to implement, test, and analyze the behavior of systems with respect to specifications.

One of those approaches is *model-based testing* [10], which is a formal approach to subject a model of an implementation to series

of tests that try to falsify the specification according to which the implementation was created, in order to find faults in the implementation. To formalize this, the concept of *input–output conformance* (IOCO) was proposed by [25].

In the IOCO testing relation, the specification provides the basis for the behavior of the implementation in that it dynamically defines the outputs that the implementation is allowed to emit. If other than the specified outputs are emitted by the implementation, it is not IOCO with respect to the specification and either the specification or the implementation need to be modified.

The reason of non-conformance has to be evaluated separately for each case. If inspection of the reason behind why the implementation is not IOCO with respect to the specification reveals that the specification is correct, the implementation needs to be modified. However, in some cases, such as when having a legacy system [26] for which the specification was designed from outdated documentation, it may in fact be the implementation that is correct and the specification that is faulty. In this case, it is the specification that needs to be modified.

To give the formal definition of IOCO, consider two disjoint sets of input actions I and output actions O. The output actions are the actions initiated by the system under test and are expressed with an exclamation mark, such as $!\,a \in O$. The input actions are commands to the system and are expressed with a question mark such as $a? \in I$. Now, we consider a labelled transition system to elaborate the concept of IOCO and give the formal definition.

Definition 1. An *I/O labelled transition system* (LTS) is a 4-tuple $\langle S, s_0, L, \rightarrow \rangle$ where:

- S is a non-empty set of states;
- $s_0 \in S$ is the initial state;
- L is a countable set of labels. These represent observable actions of a system i.e. $L = I \cup O$ where I and O are as above. Consider also a *quiescence* symbol $\delta \notin L$, and define the sets $L_\delta = L \cup \{\delta\}$ and $O_\delta = O \cup \{\delta\}$;
- $\rightarrow \subseteq S \times L_\delta \times S$ is a transition relation such that, $p \rightarrow aq$ implies $\langle p, a, q \rangle \in \rightarrow$ and $p \rightarrow a$ for $a \in L_\delta$, if there exists $q \in S$ such that $p \rightarrow aq$. Similarly, $p \not\rightarrow a$, for $a \in L_\delta$, if there exist no q such that $p \rightarrow aq$. In addition, only coherent quiescent systems are allowed, so $\rightarrow$ also satisfies the following:
  - if $p \rightarrow \delta p'$, then $p = p'$ i.e. a quiescent transition is always reflexive.
  - if $p \not\rightarrow !\,a$ for all $!\,a \in O$, then $p \rightarrow \delta p$, i.e. a state with no outputs is quiescent.
  - if $p \rightarrow !\,a$ for some $!\,a \in O$, then $p \not\rightarrow \delta$, i.e. a state with some output is not quiescent.

Furthermore, a trace t is a finite sequence of symbols of L_δ i.e. $t \in L_\delta^*$, including the empty trace ε. When the transition relation is restricted to be a function, and thus for $p \rightarrow aq$ and $p \rightarrow aq'$ it holds that $q = q'$, the resulting LTS is said to be *deterministic*.

Additional definitions needed to express the IOCO relation in Definition (6) are as follows.

Definition 2. The set of traces from a state p in an LTS is

$$\mathbf{traces}(p) = \{t \in L_\delta^* \mid p \rightarrow t\}. \tag{1}$$

For an LTS $A = \langle S, s_0, L, \rightarrow \rangle$, its set of traces are the ones defined from its initial state

$$\mathbf{traces}(A) = \mathbf{traces}(s_0). \tag{2}$$

Definition 3. The set of states reached *after* a trace t from a state p is

$$\mathbf{after}(p, t) = \{p' \in S \mid p \rightarrow tp'\}. \tag{3}$$

For an LTS $A = \langle S, s_0, L, \rightarrow \rangle$, the set of states reached *after* a trace t is

$$\mathbf{after}(A, t) = \{p' \in S \mid s_0 \rightarrow tp'\}. \tag{4}$$

For a deterministic LTS, **after**$(\cdot, \cdot)$ always returns a singleton set. Then we write **after**$(p, t) = p'$.

Definition 4. The set of *outputs* from a state p is

$$\mathbf{outs}(p) = \{!a \in O_\delta \mid p \rightarrow !a\}. \tag{5}$$

In IOCO the implementation and the specification are regarded as discrete event systems, typically modeled as finite-state machines (FSMs) [24], with the events representing the inputs and outputs. For an implementation G and specification S, the formal definition of the IOCO testing relation [27] can now be stated as.

$$\forall t \in \mathbf{traces}(S) : \mathbf{outs}(\mathbf{after}(G, t)) \subseteq \mathbf{outs}(\mathbf{after}(S, t)) \tag{6}$$

The formal IOCO definition (6) is interpreted as an implementation G conforms to a specification S, if for all the traces in the specification the output events possible from the state reached by the implementation after a trace form a subset of the possible output events from the state reached by the specification after the same trace. Whenever this subset relation between the respective sets of output events exist, the implementation is said to be IOCO with respect to the specification, for that particular trace. If the implementation is IOCO with respect to the specification for all the traces defined by the specification, then the implementation is said to be IOCO with respect to the whole specification.

As an example consider the FSM shown in Fig. 2. In the FSM, *start*, *!DoneG*, *!DoneP*, and *!DoneW* are input and output events, with the output events prefixed by an exclamation mark. The possible traces of the FSM are the empty trace of length zero, the *start* trace of length one, and the traces of length two *start.!DoneG*, *start. !DoneP*, and *start.!DoneW*. The outputs after the trace *start* are *!DoneG*, *!DoneP*, and *!DoneW*. For all other traces the possible outputs in the given example is the empty set.

For a specification that specifies that after the *start* trace only the *!DoneG* and *!DoneP* outputs are valid, an implementation looking as the FSM of Fig. 2 would not be IOCO, as the outputs from the implementation after the *start* trace would not be a subset of the outputs of the specification after the same trace. However, note that an implementation is allowed to define *fewer* output events than the specification after a trace.

The above mentioned concepts **traces**, **outs**, and **after** related to the IOCO testing relation need to be appropriately applied to their respective conceptual equivalent in the safety PLC code. The concept of **traces** is related to *sequences of operations* programmed in the PLC code that occur dynamically depending on the inputs and outputs. Similarly, **after** will be the post operation scenario in

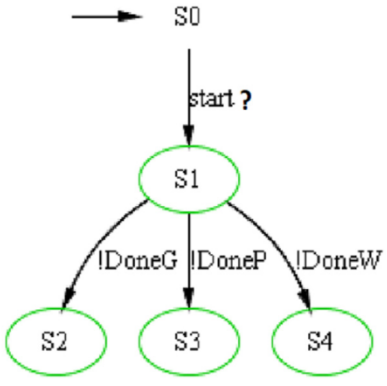

Fig. 2. FSM representing three alternative operations.

the implementation and **outs** will be the outputs from the safety PLC logic. These PLC outputs will then be compared to the expected outputs to check if the implementation is IOCO or not with respect to the specification.

In practice, Boolean signal values are shared between the safety PLC logic and the specification, not events. Thus, the notion of subsets is to be adapted to Boolean signals in the way that *False* is a subset of both *False* and *True*, while *True* is considered to be a subset of only *True*. Hence, if the specification output for a certain Boolean signal is *False* and the implementation output for the same signal is *True*, then the IOCO relation fails. The problem of combinatory explosion is not an issue for testing safety logic using IOCO, since typically, the number of inputs and outputs for a safety code is small. However, large number of inputs and outputs could lead to traceability issue if not managed properly.

5 Testing and validation of safety PLC logic in a virtual environment

In this section, the approach to test and validate the safety PLC logic in a virtual environment is detailed. The presented approach is general in nature and not brand specific. To create a simulation model, any software e.g. Delmia [13], Process Simulate [12] etc. with features to create and simulate models of physical systems and human dynamics can be used. Similarly, due to the IEC 61131-3 PLC programming standard, a PLC of any brand can be used.

Compared to the proposed approach to test and validate the safety PLC logic, as presented in Fig. 1, the actual setup differs slightly [28] as shown in Fig. 3. A single standard PLC is used to implement both the PLC logic for the nominal behavior and the safety PLC logic. Furthermore, the results of the executed tests, i.e. if the test has passed or failed, is also determined by the standard PLC instead of the test initiating software. The test initiator software, which is the HMI in the actual setup is used to initiate tests and for the visualization of test results.

The proposed setup in Fig. 1 is changed due to a problem [28] with the safety functionality that occurred during implementation. The PLC used in the implementation is a S7-1500 PLC [29] with integrated safety functions. This integrated safety functionality was unable to send and receive safety signals with the simulation model. Being unable to rectify this problem, despite numerous attempts, the proposed setup was modified. The modified setup does not have the safety PLC. Due to this, certain hardware safety functions such as redundant controllers and I/Os [29], which are associated with hardware reliability cannot be implemented and tested. However, in terms of logic development and testing, a safety PLC is the same as a standard PLC. Therefore, the modified setup does not affect the primary goal of testing safety PLC logic in a virtual environment.

The simulation model is created in the software Process Simulate [12], and a Simba Box [30] is used to establish connection with Process Simulate and the PLC. The manufacturing company provided partial CAD models of the actual cell, which were

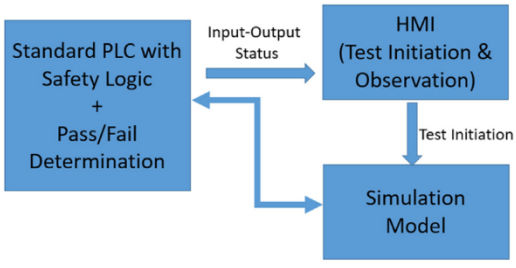

Fig. 3. Actual setup.

elaborated and enriched manually in Process Simulate by adding operations both for nominal and safety behaviour.

In the use case, different safety related scenarios are implemented in the simulation model. In addition to visual validation, the safety PLC code is validated using the IOCO testing approach.

The safety PLC code is typically interlocked with the relevant inputs and outputs of the nominal behavior. As a result, when safety critical scenarios arise, the nominal operation will be disrupted with respect to the safety PLC logic. To resume the nominal operation again, certain steps are required, e.g. resetting the emergency stop button. The nominal operations that will be disrupted in the use case are loading, picking, and placing of parts. To resume the nominal operations, the steps required will vary depending on the activated safety PLC code. If the robot has been paused due to human presence, the nominal operation can begin as soon as the human leaves the critical zone. But if the robot has been stopped due to human interference, the *Reset* button needs to be pressed manually.

In addition to the mechanical and electrical equipment, the simulation model includes a model of a human operator that can subject the system to certain scenarios e.g. opening the fence door, breaking the light barrier etc. This human operator disrupts the nominal actions, which are being carried out in the simulation model. The human model in the simulation is controlled with the *part flow operation* instead of the Jack module of Process Simulate [12], as triggering the safety sensors does not require detailed simulation of human movements.

The part flow operation is a feature in Process Simulate that allows movement of different CAD objects during simulation through pre-defined paths called *via points*. This part flow operation feature is used to move the model of a human through pre-defined via points, which were marked in four safety zones. These via points are defined using the placement manipulator function available in Process Simulate and is added to the path editor that allowed the mapping of human behavior, e.g. walking.

The disruption caused by the human is detected by the safety sensors in the simulation model, causing the sensor values to change. As a response to this change in the sensor values, the respective outputs will be triggered in the safety logic.

The outputs triggered as a result of human disruption in the safety PLC logic have tags associated with them in the human–machine-interface (HMI). These tags will not illuminate if the implementation is IOCO with respect to the specification or, in other words, if the test is passed. If the implementation is non-IOCO then the tags will illuminate and a manual amendment is required in the safety PLC code to make it IOCO. This amendment is carried out on the assumption that the specification is correct, which is the basis of the IOCO testing methodology. In this application, time constraints are not taken into account and only logical correctness will be tested.

5.1 Use case

The implementation of the IOCO concept for the validation of the safety PLC logic is carried out on the production cell illustrated in Fig. 4. This production cell is situated at a manufacturing company in Sweden. The cell contains one robot and three stations. Out of these three stations, *Station 3* is a loading station, on which a part is initially loaded, *Station 1* and *Station 2* are place stations. Initially a part arrives at the loading *Station 3*. From there the part is picked up by the robot. After being picked, it is placed on either *Station 1* or *Station 2*.

The cell is divided into four zones to cater for operations related to human safety. *Zone 1*, *Zone 2*, and the *Centre Zone* are monitored by floor scanners, while *Zone 3* is monitored by a light beam to

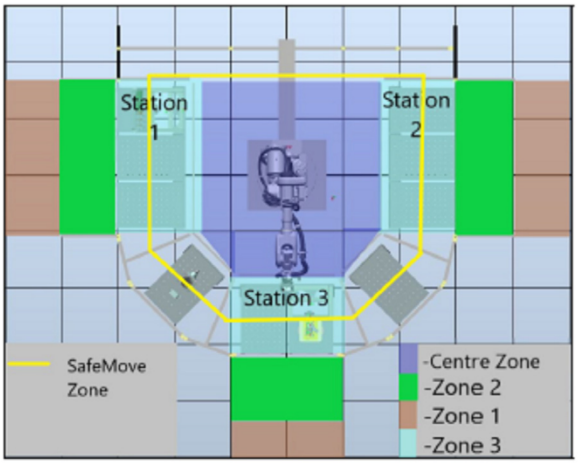

Fig. 4. Production cell.

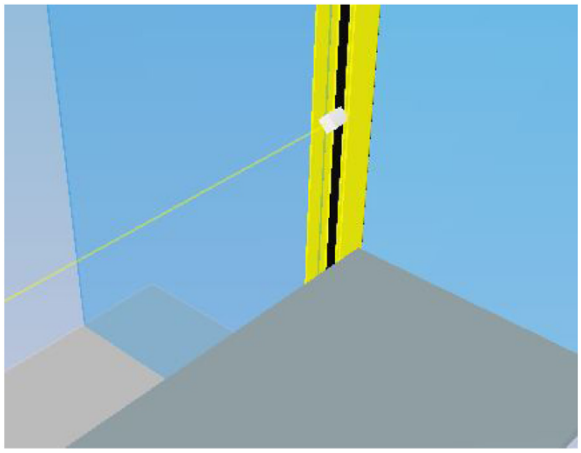

Fig. 6. Light beam.

detect human presence. For the robot operation a *SafeMove Zone* is defined, which transcends the boundary of *Zone 3* at all three stations.

The human operator included in the simulation model, after getting triggered, will cause interference in the zones. This interference is either in terms of breaking the light beam or being detected by the floor scanners. In Process Simulate, different available options are used to mimic the behavior of the respective sensors.

The floor scanners are mimicked using proximity sensors to detect objects in a defined area, which can be seen in Fig. 5 as a rectangular grey mat, which defines the detection range of the floor scanner. The light beam is created by using a light sensor from the software. Fig. 6 shows the created light beam to detect human presence.

In terms of safety logic, there are different scenarios possible when human presence is detected in the specific zones of the cell. If the human presence is detected in Zone 1, see Fig. 4, the robot will continue to carry out its task at normal speed, and nominal operation is not disrupted by the PLC safety logic. In Zone 2, the robot slows down its movement in case of human presence. However, due to unavailability in Process Simulate of functions for controlling the speed of the robot, this behaviour is not implemented for the proposed use case.

Zone 3 has two different safety scenarios associated with it. One scenario is when the robot *pauses* its movement due to human interference at any of the three work stations while the robot tries to enter the SafeMove zone of that particular station. The second scenario is when the robot *stops* its activity in the SafeMove region of any work station because the human enters that particular station.

For the Centre Zone, the robot *stops* when a human opens the fence door. The robot, after being stopped, requires a manual reset in the same way as for Zone 3.

5.1.1 Human–machine interface

The human–machine-interface (HMI) is used to initiate and monitor tests. The software used is *SIMATIC WinCC* [31]. After the execution of each test, the HMI will wait for the output signals, i.e. *Robot stop* and *Robot pause*. These two output signals are modelled in the HMI for the purpose of visualization.

The HMI, Fig. 7, contains a *Start* button to trigger the main nominal operation cycle, that is, the loading and pick-and-place operations of the robot. There are four buttons modelled to initiate different tests in the cell while the nominal operations are being carried out. Each of the four buttons that initiate tests has an indication lamp attached to it. These lamps indicate *Test status*, which will illuminate if the executed test fails.

To reset the robot after being stopped in the case that the human breaks the light beam, a *Reset* button is modelled, which is required to resume nominal operations. To exit the HMI, the exit button can be pressed.

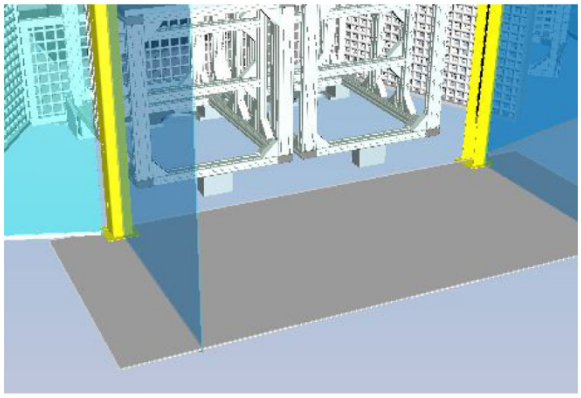

Fig. 5. Floor scanner.

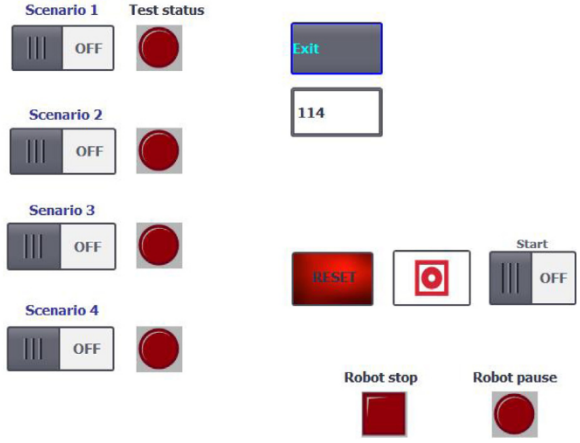

Fig. 7. Human–machine-interface.

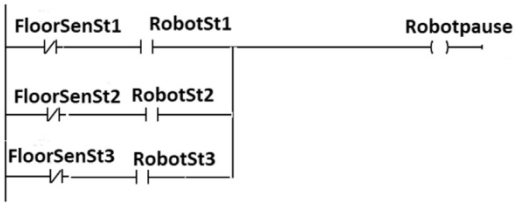

Fig. 8. Robot pause safety logic.

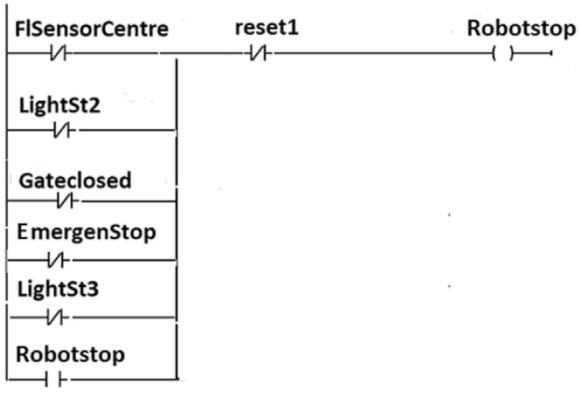

Fig. 9. Robot stop safety logic.

5.1.2 Safety PLC code

The company did not provide the actual PLC code per company policy. So, the PLC code for both nominal and safety is created manually from the specifications discussed with the manufacturing company.

The safety PLC code created for the proposed use case revolves around two safety outputs i.e. pausing and stopping of the robot. These two safety functions are tested using five different scenarios in the simulation model, which cover all the safety sensors and the logic associated with them.

Fig. 8 shows the safety PLC logic related to the robot pause function. In the safety code, the *Robotpause* output is activated via inputs *FloorSenSt1*, *FloorSenSt2*, or *FloorSenSt3* that correspond to the floor sensors installed at each station. These are activated in conjunction with the respective inputs *RobotSt1*, *RobotSt2*, and *RobotSt3*, that correspond to the presence of robot at *Station1*, *Station2*, and *Station3*, respectively.

Fig. 9 shows the safety PLC logic of the robot stop function. There are three different scenarios in which the robot is stopped. The first scenario is due to the activation of safety sensor inputs *FlSensorCentre*, *LightSt2*, or *LightSt3*. The second scenario relates to the input *EmergenStop*, which corresponds to the emergency stop button; the third scenario is related to the fence gate input *Gateclosed*, i.e. whether the fence gate is closed or not. In the safety code, the *Robotstop* output is activated via parallel combination of *FlSensorCentre*, *LightSt2*, *LightSt3*, *EmergenStop*, or *Gateclosed* in conjunction with the *reset1* input.

The default values of the safety sensors, emergency stop button etc. received from the simulation model are normally high in both robot pause and the robot stop safety logic. These safety sensor signals are represented using normally closed contacts in the safety logic. This combination of normally high signals and the normally closed contacts in the safety logic makes the *Robotpause* and the *Robotstop* outputs low under nominal conditions, hence the robot remains operational. However, in case of human interference, the received sensor signals goes low, which makes the *Robotpause* and the *Robotstop* signals go high, which in turn makes the physical robot pause or stop.

Table 1
Traces in robot pause safety logic.

Traces	*Outs*
FloorSenSt1. RobotSt1	*Robotpause*
FloorSenSt2. RobotSt2	*Robotpause*
FloorSenSt3. RobotSt3	*Robotpause*

Table 2
Traces in robot stop safety logic.

Traces	*Outs*
FlSensorCentre. reset1	*Robotstop*
LightSt2. reset1	*Robotstop*
LightSt3. reset1	*Robotstop*
Gateclosed. reset1	*Robotstop*
EmergenStop. reset1	*Robotstop*

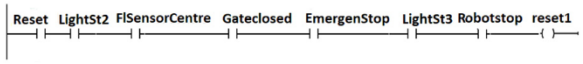

Fig. 10. Reset logic.

From the perspective of IOCO, there are many traces possible due to the combinatorial nature of the logic. However, all combinations eventually activate either the *Robotpause* or the *Robotstop* output. Therefore, only parallel traces in the safety PLC logic are highlighted in Tables 1 and 2.

The normally open contact *Robotstop* in the robot stop logic is not regarded as an input event, because it is only used in the safety PLC code to hold the robot in the stopped state. The *reset1* input on the other hand is connected to the reset logic. This *reset1* input is a normally closed contact and works inversely when the *Reset* button is activated.

The *Reset* input shown in Fig. 10 is provided to manually reset the PLC code, which is activated to resume operation via the HMI *Reset* button. The reset logic is interlocked with inputs *FlSensorCentre, LightSt2, LightSt3, EmergenStop*, and *Gateclosed*, so that the robot can only resume operation, once the intruded zone has been cleared from human presence. The input *Robotstop* is added to hold the robot in the stopped state, until the *Reset* is pressed.

5.1.3 Testing and validation

The tests created can be seen in Fig. 11, each test's result after being initiated is determined by the *Test determination Logic* illustrated in Fig. 12. Tests are initiated via the HMI (Fig. 7) by pressing the buttons corresponding to the specific test, these buttons are designated *Scenario1*, *Scenario2*, *Scenario3*, and *Scenario4*. After test initiation, if the test status indication stays *low*, this signals a pass for the executed test. The inputs *Scenario1, Scenario2, Scenario3*, and *Scenario4* in Fig. 11 instigates a model of a human via the outputs *Human1*, *Human2*, *Human3*, and *Human4*, which works in conjunction with *human1end*, *human2end*, *human3end*, and *human4end* in a holding coil PLC structure.

Test one instigates a human to interfere with the nominal operation at *Station3*. According to the safety logic in Fig. 8, the inputs *FloorSenSt3* and *RobotSt3* pause the robot as the human enters *Station3* in the presence of the robot. The *Robotpause* lamp in the HMI illuminates as a response to this interference. At the same time, the output *test1fail* in the test result determination logic in Fig. 12 stays *low*, which constitutes a pass for the executed test.

In the second test, a human breaks the light beam at *Station3* in the presence of the robot at *Station3*. According to the safety logic

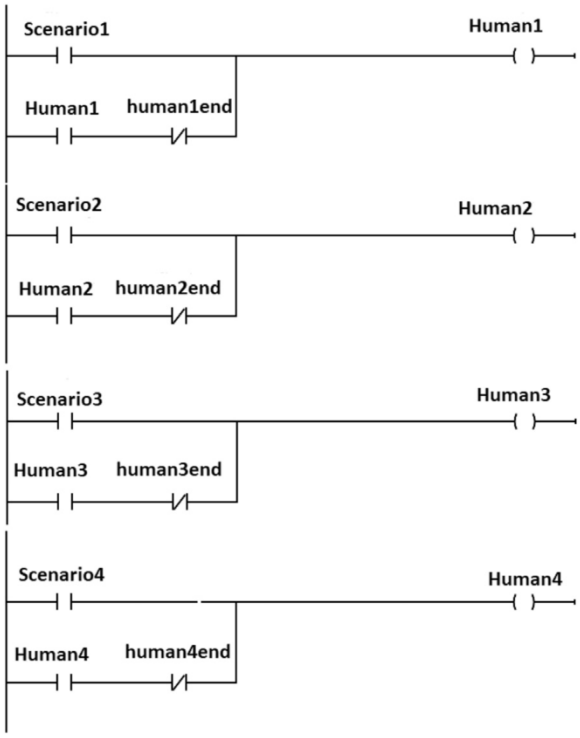

Fig. 11. Tests.

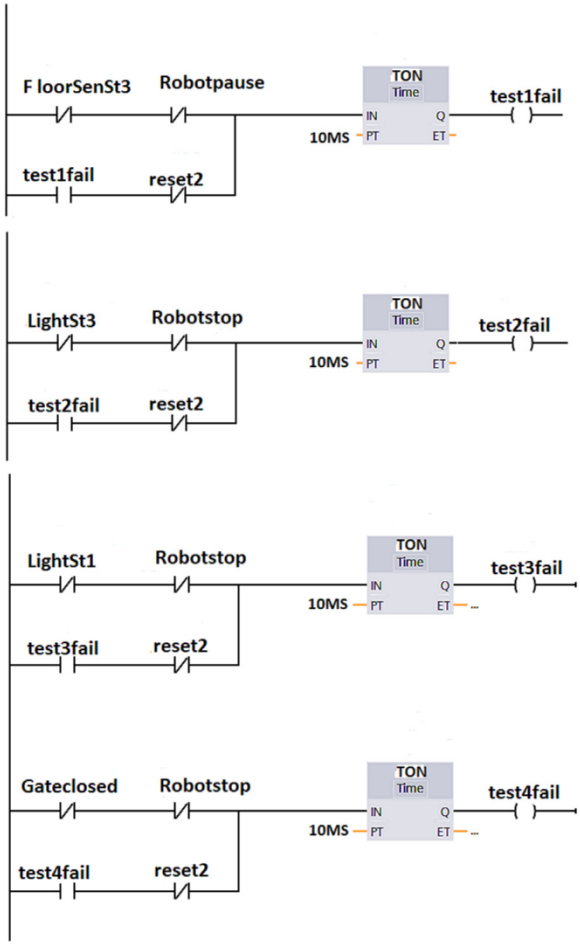

Fig. 12. Test results determination.

in Fig. 9, the input *LightSt3*, which is related to the light beam installed at *Station3* stops the robot. As a response to this event, the *Robotstop* is illuminated in the HMI, hence the input–output conformance relation for test two is validated, and the *test2fail* output stays *low*.

In the third test, the human breaks the light beam installed at *Station1*, but the robot does not stop. Due to this failure in the safety logic, the test status lamp of the third test turns *red* in the HMI, indicating that the initiated test has failed, hence the input–output conformance relation fails. Upon inspection of the safety PLC logic shown in Fig. 9, it is found that the input related to the light beam sensor, which is *LightSt1*, is not taken into account by the code. This error in the safety PLC code was fixed manually after the failure of the test by adding the required input *LightSt1*.

In the fourth test, a human interferes in the *CentreZone* by opening the fence gate, while the nominal operation is being carried out. This disruption causes the robot to stop as per the safety PLC logic shown in Fig. 9. The *Robotstop* is triggered via the *Gateclosed* input as the fence gate opens, hence the input–output conformance relation is validated for scenario four.

Fig. 12 represents the test results determination logic for the four tests mentioned above. Each rung in this logic determines the status (pass or fail) of the executed test with a normally closed contact of the triggered safety sensor e.g. *FloorSenSt3*, *LightSt3*, etc. in conjunction with the normally closed contact of either the *Robotpause* or *RobotStop* output. For each test, the failure is highlighted in the HMI indications, which are connected to the outputs *test1fail*, *test2fail* etc. These outputs are also used in a holding coil configuration with an input *reset2*. The *reset2* input is provided to reset the executed test to its initial state, while the timers are provided to have some tolerance for signal flickering that can occur during human interference.

Apart from the mentioned tests, other scenarios were also tested for the given use case, which were based on activation of multiple sensors simultaneously in different zones. However, the robot is either paused or stopped, which was trivial and was already tested in the four tests mentioned above. Therefore, these tests are not detailed in this paper.

The errors found in the safety PLC logic during testing of the other scenarios, which are not detailed in the paper were fixed manually after inspection, so that the safety PLC code conforms to the required specification. Hence, the applied testing approach using the IOCO testing relation showed its viability on a real industrial system, as it helped in finding errors in the safety PLC code in the virtual environment.

6 Conclusion

In this paper, an approach to test and validate safety PLC code based on the IOCO testing relation in a virtual environment is detailed. Similar to virtual commissioning, in the proposed approach, the safety PLC code is tested using a simulation model of a physical system. The proposed approach is applied on a use case to demonstrate how input–output conformance testing can be used for the validation of safety PLC logic. The outcome of the testing shows the viability of the approach in an industrial setting, as the errors found in the safety PLC code using the simulation model can be fixed before the factory acceptance phase. Due to this, the engineers can prepare themselves better before the actual factory acceptance testing phase and is expected to spend less time at the contractors facility, which as a result may help in further reduction in the physical commissioning time of a production system. In this work, the errors found in the safety PLC code were fixed manually but in the future, this work will be extended in terms of using

automatic adjustment of non-conforming implementation with respect to the specification. Furthermore, in the future, QuickCheck [32] will be used to generate more complex test cases.

References

[1] Liu, Z., Diedrich, C., Suchold, N., 2012, Virtual Commissioning of Automated Systems. INTECH Open Access Publisherhttps://www.intechopen.com/books/automation/virtualcommissioningofautomated-systems.

[2] Z. Liu, C. Diedrich, N. Suchold, https://www.intechopen.com/books/automation/virtualcommissioningofautomated-systems, Virtual Commissioning of Automated Systems, INTECH Open Access Publisher, 2012.

[3] ABB, ABB Reduces Project Cost for Drives Through Virtual Commissioning by 25 percent, http://www.abb.com/cawp/seitp202/0f58dfe7b53b6829c12581d90064cf06.aspx.

[4] Hoffmann, P., Schumann, R., Maksoud, T.M., Premier, G.C., 2010, Virtual Commissioning of Manufacturing Systems a Review and New Approaches for Simplification. 24th European Conference on Modelling and Simulation (ECMS 2010), 175–181. 10.1.1.463.7252.

[5] Mathias, O., Gerrit, W., Oliver, D., Benjamin, L., Markus, S., Leon, U., 2014, Automatic Model Generation for Virtual Commissioning Based on Plant Engineering Data. IFAC Proceedings, 47/3: 11635–11640In: *https://www.sciencedirect.com/science/article/pii/S1474667016434671*.

[6] Seidel, S., Donath, U., Haufe, J., 2012, Towards an Integrated Simulation and Virtual Commissioning Environment for Controls of Material Handling Systems. Proceedings of the Winter Simulation Conference Winter Simulation Conference, 252In: *https://ieeexplore.ieee.org/document/6465081*.

[7] Dahl, M., Bengtsson, K., Bergagård, P., Fabian, M., Falkman, P., 2016, Integrated Virtual Preparation and Commissioning: Supporting Formal Methods During Automation Systems Development. IFAC-PapersOnLine, 49/12: 1939–1944In: *https://www.sciencedirect.com/science/article/pii/S2405896316312022*.

[8] L. Fransén, National Electric Vehicle Sweden, Manufacturing Engineer, Private Conversation.

[9] Ljungkrantz, O., Akesson, K., Yuan, C., Fabian, M., 2012, Towards Industrial Formal Specification of Programmable Safety Systems. IEEE Transactions on Control Systems Technology, 20/6: 1567–1574. 10.1109/TCST.2011.2169262.

[10] Utting, M., Legeard, B., 2007, Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. http://dx.doi.org/10.1016/B978-0-12-372501-1.X5000-5.

[11] Mikucionis, M., Larsen, K.G., Nielsen, B., 2004, T-UPPAAL: Online Model-Based Testing of Real-Time Systems. in: Proceedings of the 19th IEEE International Conference on Automated Software Engineering (IEEE Computer Society), pp.396–397. 10.1109/ASE.2004.1342774.

[12] Siemens, Process Simulate, https://www.plm.automation.siemens.com.

[13] ELMIA, Digital Manufacturing Software, https://www.3ds.com/products-services/delmia/.

[14] Lee, C.G., Park, S.C., 2014, Survey on the Virtual Commissioning of Manufacturing Systems. Journal of Computational Design and Engineering, 1/3: 213–222. 10.7315/JCDE.2014021.

[15] Winther, S., 2017, Virtual Commissioning of Production Process, Master's thesis.In: *http://studentarbeten.chalmers.se/publication/250446virtualcommissioningofproductionprocess*.

[16] Dominka, S., Schiller, F., Kain, S., 2007, Hybrid Commissioning-Speeding-Up Commissioning of Field Bus Driven Production Plants. ICM2007 4th IEEE International Conference on Mechatronics (IEEE), pp.1–6. 10.1109/ICMECH.2007.4280024.

[17] Kain, S., Schiller, F., Dominka, S., 2011, Methodology for reusing real-Time Hil Simulation Models in the Commissioning and Operation Phase of Industrial Production Plants. Edited by Ganesh R Naik, 143. 10.5772/14673.

[18] Canet, G., Couffin, S., Lesage, J.-J., Petit, A., Schnoebelen, P., 2000, Towards the Automatic Verification of PLC Programs Written in Instruction List. 2000 IEEE International Conference on Systems, Man, and Cybernetics, Vol. 4 (IEEE), pp.2449–2454. 10.1109/ICSMC.2000.884359.

[19] Provost, J., Roussel, J.-M., Faure, J.-M., 2011, Translating Grafcet Specifications into Mealy Machines for Conformance Test Purposes. Control Engineering Practice, 19/9: 947–957. http://dx.doi.org/10.1016/j.conengprac.2010.10.001.

[20] Perin, M., Faure, J.-M., 2013, Building Meaningful Timed Models of Closed-Loop DES for Verification Purposes. Control Engineering Practice, 21/11: 1620–1639. 10.3182/20090603-3-RU-2001.00159.

[21] Riera, B., Benlorhfar, R., Annebicque, D., Gellot, F., Vigario, B., 2011, Robust Control Filter for Manufacturing Systems: Application to PLC Training. IFAC Proceedings, 44/1: 14265–14270. 10.3182/20110828-6-IT-1002.01976.

[22] Frey, G., Litz, L., 2000, Formal Methods in PLC Programming. 2000 IEEE International Conference on Systems, Man, and Cybernetics, Vol. 4 (IEEE), pp.2431–2436. 10.1109/ICSMC.2000.884356.

[23] Khan, A., Falkman, P., Fabian, M., 2017, Virtual Engineering Framework for Automatic Generation of Control Logic Including Safety. 2017 13th IEEE Conference Automation Science and Engineering (CASE) (IEEE), pp.648–653. 10.1109/COASE.2017.8256176..

[24] Cassandras, C., Lafortune, S., 2009, Introduction to Discrete Event Systems, SpringerLink Engineering. Springer US. 10.1007/978-0-387-68612-7.

[25] Tretmans, G., 1996, Test Generation with Inputs, Outputs and Repetitive Quiescence.46In: *https://research.utwente.nl/en/publications/test-generation-with-inputs-outputs-and-repetitive-quiescence-2*.

[26] Khan, A., Falkman, P., Fabian, M., 2018, Digital Twin for Legacy Systems: Simulation Model Testing and Validation. 2018 14th IEEE Conference Automation Science and Engineering (CASE) (IEEE), . 10.1109/COASE.2018.8560338.

[27] Gregorio-Rodríguez, C., Llana, L., Martínez-Torres, R., 2013, Input–Output Conformance Simulation (IOCOS) for Model Based Testing. Formal Techniques for Distributed Systems (Springer), pp.114–129. 10.1007/978-3-642-38592-69.

[28] Swathanandan, J., Kristoffer, G., 2018, Virtual Commissioning for Safety Verification of a Collaborative Robotic Cell, Master's thesis..

[29] Siemens, Simatic-s7-1500 plc, https://www.siemens.com/global/en/home/products/automation/systems/industrial/plc/simatic-s7-1500html.

[30] Siemens, Simba pnio hardware platform. https://cache.industry.siemens.com/dl/files/344/109475344/att926827/v1/HelpEN.pdf.

[31] Siemens, Wincc, https://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/simatic-wincc/pages/default.aspx.

[32] Claessen, K., Hughes, J., 2011, Quickcheck: A Lightweight Tool for Random Testing of Haskell Programs. ACM Sigplan Notices, 46/4: 53–64.